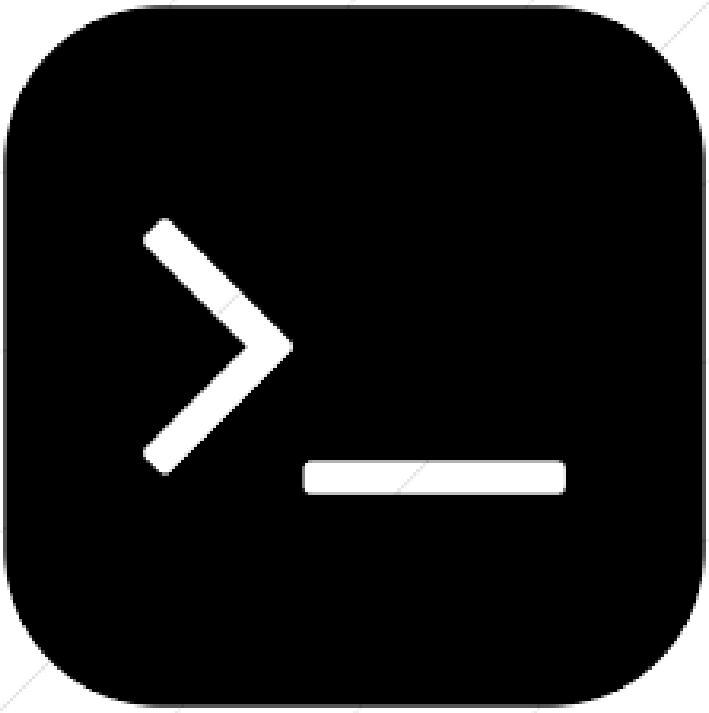

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Εργασία 2018-2019

ΖΗΤΟΥΜΕΝΟ

Το ζητούμενο της εργασίας ήταν η δημιουργία ενός shell το οποίο θα μπορεί να δουλεύει είτε σε interactive mode είτε σε batch mode, μέσω της χρήσης της γλώσσας C.

Παυλίδης Μιχαήλ
Ιάσων - 9015

Εισαγωγή

Το shell το οποίο κληθήκαμε να δημιουργήσουμε θα έπρεπε να υποστηρίζει δύο λειτουργίες. Μια άμεση, στην οποία εισάγουμε εμείς τις εντολές που θέλουμε να τρέξουμε και μια έμμεση κατά την οποία το πρόγραμμα διαβάζει τις εντολές μέσω ενός αρχείου. Το λεγόμενο *batchfile*. Θα αναφερθώ ξεχωριστά στις δύο αυτές κατηγορίες και θα προσπαθήσω να αναφέρω τις διαφορές και τις δυσκολίες που συνάντησα.

Η υλοποίηση μου υποστηρίζει τα ακόλουθα χαρακτηριστικά:

- Απλές εντολές
- Πολλαπλές εντολές διαχωρισμένες με κατάλληλους χαρακτήρες (“;” Και “&&”)
- Redirections

Επιπλέον, έχω προσθέσει αρκετούς ελέγχους για να αποφεύγω εσφαλμένες εισόδους αλλά και ιδιαίτερες περιπτώσεις. Θα αναφερθώ σε αυτό λεπτομερέστερα αργότερα.

Βασικές Συναρτήσεις

Για την καλύτερη οργάνωση του κώδικα μου δημιούργησα κάποιες συναρτήσεις τις οποίες χρησιμοποιώ κατά κόρον. Θα εξηγήσω τη λειτουργία της κάθε μιας. Έχω προσθέσει αρκετή τεκμηρίωση εντός του κώδικα ώστε να καταλάβει κάποιος εύκολα το πως δουλεύουν. Οι βασικότερες είναι οι ακόλουθες:

- **Split():** Πρόκειται ίσως για τη συνάρτηση που χρησιμοποιείται περισσότερο από κάθε άλλη. Σκοπός της είναι ο διαχωρισμός ενός *string* με βάση κάποιο *delimiter* και η αποθήκευση των κομματιών σε ένα πίνακα.

- **Parse():** Είναι η συνάρτηση η οποία επεξεργάζεται την είσοδο του προγράμματος. Ελέγχει αν έχουμε απλή εντολή ή πολλαπλές, αν υπάρχει κάποιο συντακτικό σφάλμα καθώς και αν εμπεριέχεται pipe. Αν η είσοδος περάσει τους ελέγχους καλείται η *split()* για τον κατάλληλο διαχωρισμό και στη συνέχεια εκτελούνται οι εντολές μέσω της επόμενης συνάρτησης.

- **Execute():** Είναι η πιο περίπλοκη συνάρτηση. Δημιουργεί μια διεργασία-παιδί μέσω της συνάρτησης *fork()* και αρχικά ελέγχει αν απαιτείται κάποιου είδους redirection. Αν ναι, δημιουργεί ή ανοίγει τα κατάλληλα αρχεία και κάνει *redirect* την είσοδο ή την έξοδο. Στη συνέχεια εκτελεί την εντολή που θέλουμε και αν αποτύχει, είναι υπεύθυνη για την σωστή έξοδο από το *process* που έχει δημιουργηθεί.

- **TrimSpace():** Είναι βοηθητική συνάρτηση και χρησιμοποιείται για να αποκόπτει από τα άκρα ενός *string* χαρακτήρες όπως τα κενά, οι αλλαγές γραμμής κλπ (white-spaces). Είναι απλή αλλά πολλή χρήσιμη καθώς μέσω της χρήσης αυτής δεν υπάρχει πρόβλημα αν δοθεί μια εντολή με πολλά κενά.

- **IsEmpty():** Και αυτή είναι βοηθητική συνάρτηση και απλά ελέγχει αν η είσοδος που δόθηκε ή διαβάστηκε είναι κενή γραμμή.

Interactive Mode

Η ροή του προγράμματος στο interactive mode είναι η ακόλουθη:

1. Αρχικά, ζητείται η είσοδος. Εφόσον αυτή δοθεί, γίνεται έλεγχος για το αν αυτή είναι κενή ή η λέξη “quit”. Σε περίπτωση που δεν ισχύει κάτι από τα δύο, καλείται η *parse().*
2. Η *parse()* όπως είπαμε και παραπάνω ελέγχει για το σωστό συντακτικό των εντολών και αναλόγως την περίπτωση εκτελεί ορισμένες λειτουργίες.
3. Έπειτα, η *execute()* αναλαμβάνει την εκτέλεση των εντολών και ενημερώνει αν προέκυψε κάποιο πρόβλημα.

Batch Mode

Η ροή του προγράμματος στο batch mode είναι ίδια, με τη μόνη διαφορά να εμφανίζεται στον τρόπο με τον οποίο εισάγονται οι εντολές. Αυτήν τη φορά διαβάζεται ένα αρχείο, το όνομα του οποίου αποτελεί παράμετρο του εκτελέσιμου, και μέσα από αυτό προκύπτουν οι εντολές. Γίνεται έλεγχος για την περίπτωση των κενών γραμμών καθώς και των τμημάτων που αποτελούν σχόλια.

Δυσκολίες

Στη συνέχεια θα ήθελα να αναφερθώ στα ζητούμενα τα οποία μου φάνηκαν πιο δύσκολα και τον τρόπο με τον οποίο τα χειρίστηκα.

Εκτέλεση πολλαπλών εντολών

Ένα από αυτά ήταν η μη εκτέλεση κάποιων εντολών στην περίπτωση της εισόδου με πολλαπλές εντολές. Πιο συγκεκριμένα, όταν έχουμε το διαχωριστικό χαρακτήρα ";", η εκτέλεση των εντολών θέλουμε να συνεχιστεί ακόμη και αν κάποια από τις προηγούμενες δεν τα καταφέρει. Στην περίπτωση που έχουμε το διαχωριστικό "&&", η εκτέλεση θέλουμε να σταματήσει στο πρώτο σφάλμα.

Για να το χειριστώ αυτό χρησιμοποίησα το όρισμα της συνάρτησης *_exit(int status)*. Κάθε φορά που κάποια εντολή δεν μπορεί να εκτελεστεί, η διαδικασία στην οποία ανήκει τερματίζεται μέσω της παραπάνω συνάρτησης. Εγώ περνάω σαν όρισμα την τιμή 10. Η μητρική διεργασία καλεί τη συνάρτηση *wait(int* status)* και ουσιαστικά διαβάζει το σήμα που στέλνει η *_exit()*. Διαιρώντας αυτήν την τιμή με το 255, προκύπτει ο αριθμός που θέλουμε. Στην περίπτωση λοιπόν του διαχωριστικού "&&" ελέγχω αν το flag έγινε 10 και αν συνέβη αυτό, δεν συνεχίζω την εκτέλεση των υπόλοιπων εντολών.

Batch mode “bug”

Το πιο δύσκολο πρόβλημα που συνάντησα ήταν η διαφορετική συμπεριφορά του *interactive* και *batch* mode στην περίπτωση εσφαλμένης εντολής. Ενώ το πρώτο δούλευε σωστά, το δεύτερο παρουσιάζει διάφορα προβλήματα.

Τελικά το πρόβλημα ήταν η χρήση της συνάρτησης *exit()* αντί της *_exit()* για τον τερματισμό της διαδικασίας. Η επίλυση ήταν εύκολη αλλά ο εντοπισμός αρκετά δύσκολος.

Παρατηρήσεις

1. Ο κώδικας μου έτρεξε επιτυχώς το *test.sh* που μας δόθηκε ενώ ανταπεξήλθε σωστά σε όσους ελέγχους έκανα.
2. Τις βοηθητικές συναρτήσεις *is_empty()* και *trim_space()* τις βρήκα στο Stack Overflow οπότε υπάρχει περίπτωση να είναι κοινές με κάποιου συμφοιτητή μου.
3. Το shell μου δεν υποστηρίζει pipes και built-ins εντολές όπως η cd. Από όσο γνωρίζω όμως, δεν ζητήθηκε κάτι τέτοιο.

Επικοινωνία

michailpg@ece.auth.gr